

1 Parallel Processing Letters
2 2150013 (21 pages)
3 © World Scientific Publishing Company
5 DOI: 10.1142/S0129626421500134



Optimizing Data Intensive Flows for Networks on Chips

Junwei Zhang*

*Department of Applied Mathematics and Statistics,
Stony Brook University, Stony Brook, NY, 11794
junweizhang23@gmail.com*

Li Shi

*Department of Electrical and Computer Engineering,
Stony Brook University, Stony Brook, NY, 11794*

Yang Liu

*Department of Electrical and Computer Engineering,
Stony Brook University, Stony Brook, NY, 11794*

Thomas G. Robertazzi

*Department of Electrical and Computer Engineering and
Department of Applied Mathematics and Statistics,
Stony Brook University, Stony Brook, NY, 11794*

Received 31 March 2020

Accepted 28 April 2021

Published

Communicated by

ABSTRACT

A novel framework is proposed to find efficient data intensive flow distributions on Networks on Chip (NoC). Voronoi diagram techniques are used to divide a NoC array of homogeneous processors and links into clusters. A new mathematical tool, named the flow matrix, is proposed to find the optimal flow distribution for individual clusters. Individual flow distributions on clusters are reconciled to be more evenly distributed. This leads to an efficient makespan and a significant savings in the number of cores actually used. The approach here is described in terms of a mesh interconnection but is suitable for other interconnection topologies.

Keywords: Divisible load theory; Voronoi diagram; multi-source; Network on Chip (NOC); mesh; data intensive load; load injection.

*Corresponding author.

J. Zhang et al.

1. Introduction

The mapping of tasks on a network of processors heavily impacts the performance of parallel applications running on such a multi-processor system. A crucial task scheduling problem, utilizing the maximum benefits of parallel computing system, is considered. Scheduling multi-source divisible loads in a Network on Chip (NoC) is a challenging task as different sources should cooperate and share their computing power with others to balance their workloads and in a manner of minimizing total computational time (makespan).

In this paper a new approach is proposed to find efficient data flows in Networks on Chip (NoC). Voronoi diagram methods are used to segment a NoC homogeneous array of processors and links into clusters of processors and links. The optimal flow distribution for individual clusters operating under a given scheduling policy and set of assumptions is found using a new linear mathematical tool called the flow matrix. Individual flow distributions for each cluster are refined to make a relatively even distribution of flows across the clusters. This minimizes the negative influence of “bottleneck” clusters. This heuristic leads to an efficient value of makespan and a significant savings in the actual number of processors used (leading to significant chip number savings: about 30% in this study). This overall approach is described for a mesh interconnection of processors but is readily extended to other interconnection topologies (a toroidal network example is presented). Related work is described in the rest of this section. Section 2 presents definitions and assumptions. Section 3 discusses three models in order of increasing generality and together with examples. Section 4 examines thermal management issues. Section 5 discusses some load sharing aspects. Section 6 is the conclusion.

1.1. Related work

1.1.1. Networks on Chip (NoC)

As the number of cores increase on a single chip, conventional bus technology can no longer satisfy today’s requirements of throughput and latency. Confronted with the pressing demand for extremely high bandwidth and low power consumption, network-on-chip (NoC) architectures are proposed as a new paradigm to interconnect a large number of processing cores at the chip level. In today’s and future electronic technology, Systems-on-chip (SoC) technology has become important and even essential [1], and has emerged as a communication backbone to enable a high degree of integration in multi-core Systems-on-chip (SoC) [2–4].

Networks on chip (NoC) represents the smallest networks that have been implemented to date [1]. A popular choice for the interconnection network on such networks on chip is the rectangular mesh (Fig. 1). It is straightforward to implement and is a natural choice for a planar chip layout. Data to be processed can be inserted into the chip at one or more so-called “injection points”, that is a node(s) in the mesh that forwards the data to other nodes. Beyond NoCs, injecting data

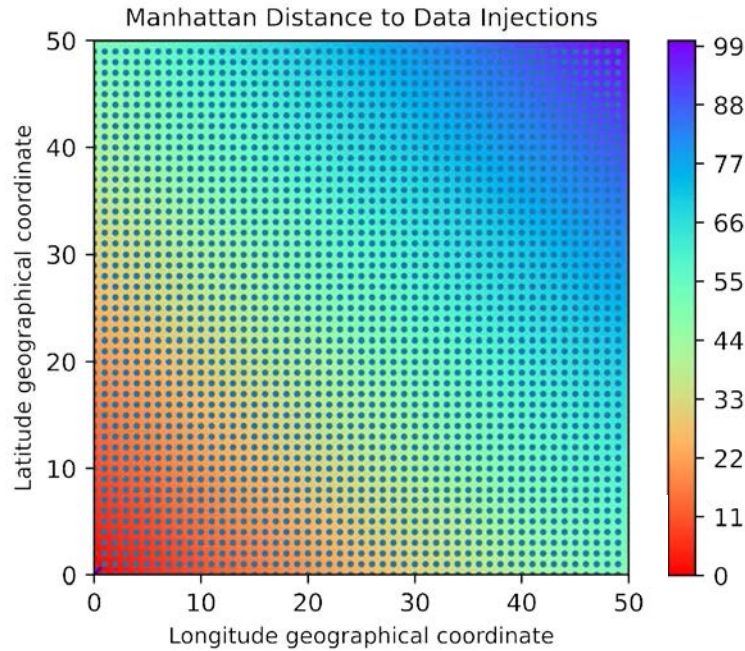


Fig. 1. A $m \times n$ mesh network ($m = 50$, $n = 50$). Data injector at $(0,0)$.

into a parallel processor's interconnection network has been done for some time, for instance in IBM's Bluegene machines [5]. In this paper, it is sought to determine, for multi-source injection points on a homogeneous rectangular mesh, how to efficiently assign a load to different processors/links in a known timed pattern so as to process a load of data in a minimal amount of time (i.e. minimizing makespan). In this paper, we succeed in presenting an efficient technique for multi-source injection in homogeneous meshes that involves no more complexity than linear equation solution. The methodology presented here can be applied to a variety of interconnection networks and switching/scheduling protocols besides those directly covered in this paper.

1.1.2. Divisible load theory

There are massive divisible load scheduling theory's applications, such as, [6–9]. Developed over the past few decades, divisible load theory assumes that load is a continuous variable that can be arbitrarily partitioned among processors and links in a network. We utilize the divisible load scheduling's optimality principle [7, 10] — makespan is minimized when one forces all processors to stop at the same time (intuitively otherwise one could transfer load from busy to idle processors to achieve a better solution). This leads to a series of chained linear flow and processing equations that can be solved by linear equation techniques, often yielding recursive

J. Zhang et al.

1 solutions for optimal load fractions and even closed form solutions for quantities
2 such as makespan and speedup.

3 1.1.3. *Multi-source assignment*

4 Wong, Yu, Veeravalli and Robertazzi [11] examined two sources grid scheduling with
5 memory capacity constraints. Marchal, Yang, Casanova, and Robert [12] studied
6 the use of linear programming to maximize throughput for large grids with multiple
7 loads/sources. Lammie and Robertazzi [13] presented a numerical solution for a lin-
8 ear daisy chain network with load originating at both ends of the chain. Finally, Yu
9 and Robertazzi examined mathematical programming solutions and flow structure
10 in multi-source problems [14].

11 1.1.4. *Voronoi diagram*

12 Voronoi diagrams are used in this paper for multi-sources assignment. Voronoi
13 diagrams are induced by a set of points (called sites or seeds): subdivision of the
14 plane where the faces correspond to the regions where one site is closest. Voronoi
15 diagrams are extensively utilized in network optimization application [15–17].

16 1.1.5. *Processor equivalence*

17 The concept of processor equivalence was first proposed in [18]. A linear daisy chain
18 of processors where processor load is divisible and shared among the processors is
19 examined. It is shown that two or more processors can be collapsed into a single
20 equivalent processor. We propose a flow matrix closed-form equation to present
21 the equivalence for single source assignment, which allows a characterization of the
22 nature of minimal time solution and a simple method to determine when and how
23 much load to distribute to processors. These works [18, 19] inspire us to adopt the
24 processor equivalence model to tackle multi-source workload scheduling problems.

25 1.2. *Our contribution*

- 26 (1) With the objective of minimizing the makespan, we propose a novel algorithm
27 framework Reduced Manhattan Distance Voronoi Diagram Algorithm (RMD-
28 VDA) to address the multi-source problem. This framework can be directly
29 extended from typical mesh and torus networks to a general graph design of
30 NoC patterns.
- 31 (2) We propose a novel mathematics tool, the flow matrix [20, 21], to calculate the
32 data fraction allocated to each processor in each cluster and the speedup of
33 each cluster.
- 34 (3) Via the extensive random simulation experiments, we demonstrate the method
35 reduces the computational core number by 30% while achieving the same
36 makespan.

2. Definitions and Assumption

A mesh network is illustrated in Fig. 1. The rainbow color bar means the Manhattan distance to data injector $(0, 0)$. For example, the $(0, 0)$ to $(0, 0)$ Manhattan distance is 0 and the color is red and the furthest point coordinate is $(50, 50)$ and the color is blue.

The following assumptions are used throughout the paper:

- For simplicity, return communication is not considered.
- The communication delay which is proportional to data size is considered.
- The time delay of computation is proportional to the data size.
- The network environment is homogeneous, that is, all the processors have the same computation capacity and the link speed between any two connected cores is identical.
- The load of data injected from each data source is identical in size.

2.1. Notations

The following notations and definitions are utilized:

- m : The number of the x-coordinate cores.
- n : The number of the y-coordinate cores.
- k : The number of data injection nodes.
- r : The rank of flow matrix [20].
- D_i : The minimum number of hops from the processor P_i to the data injectors.
- α_0 : The load fraction assigned to the root processor.
- α_i : The load fraction assigned to the i th processor.
- $\hat{\alpha}_i$: The load fraction assigned to the i th Voronoi cell.
- $\hat{\alpha}_{i,j}$: The data fraction of each processor on the l_j layer of i th Voronoi cell.
- ω : The inverse computing speed of a processor.
- ω_{eq} : The inverse computing speed on an equivalent node collapsed from a cluster of processors.
- L : The number of layers in a cell.
- z : The inverse link speed of a link.
- T_{cp} : Computing intensity constant. The entire load is processed in time $\omega \times T_{cp}$ seconds on the i th processor.
- T_{cm} : Communication intensity constant. The entire load is transmitted in time $z \times T_{cm}$ seconds over the i th link.
- T_f : The makespan for the entire divisible load solved on the root processor.
- \hat{T}_f : The finish time of the whole processor network. Here \hat{T}_f is equal to $\omega_{eq} T_{cp}$.
- $\hat{T}_{f,k}$: The finish time of k th cell.
- $T_{f,i}$: The finish time for the i th processor, $i \in 0 \cdots (m * n - 1)$.
- $\sigma = \frac{z T_{cm}}{\omega T_{cp}}$: The ratio between the communication speed to the computation speed, $0 < \sigma < 1$ [7, 22].

J. Zhang et al.

- 1 • Sp_i : The speedup of i th Voronoi cell $i \in 0 \cdots (k-1)$.
- 2 • $Speedup = \frac{T_f}{T_f} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0}$

3 2.2. Flow matrix

4 The flow matrix is the matrix form of a group of divisible load theory equations,
 5 where each equation describes the relationship between the computation time,
 6 transmission time and finish time. It can be obtained via [20, 21]. The flow matrix
 7 concept is illustrated in terms of a simple example. The load L is assigned to the
 8 corner processor P_0 , as shown in Figure 2. The whole load is processed by four
 9 processors P_0, P_1, P_2, P_3 together. The processor P_0, P_1 and P_2 start to process
 10 their respective load fraction at the same time. This includes P_1 and P_2 as they
 11 are relayed load in virtual cut-through mode at $t = 0$. Because we assume a homo-
 12 geneous network (in processing speed and communication speed), $\alpha_1 = \alpha_2$ and P_1
 13 and P_2 stop processing at the same time. The processor P_3 starts to compute when
 14 the α_1 and α_2 complete transmission. That is, the link $0-1$ and $0-2$ are occupied
 15 transmitting load to processor 1 and 2, respectively and only transmit to 3 when
 16 that is finished.

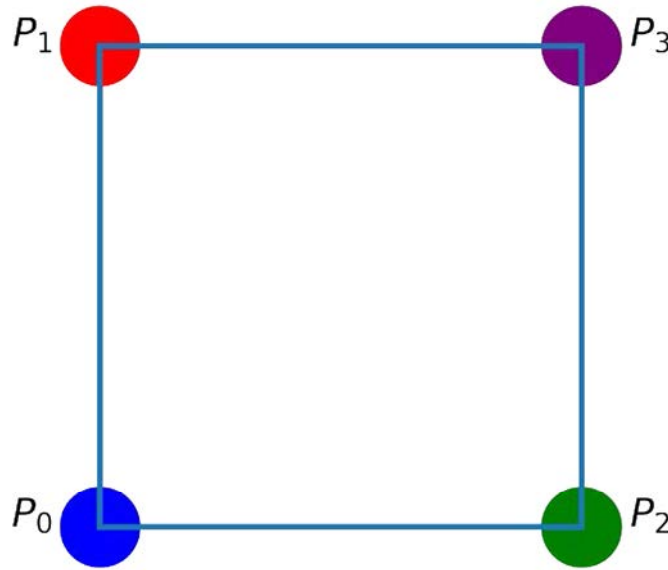


Fig. 2. The 2*2 mesh network and the root processor at $P_0(0,0)$.

17 According to the divisible load theory [6], we obtain the Gantt-like timing dia-
 18 gram Figure 3. Let's assume that all processors stop computing at the same time
 in order to minimize the makespan [10]. We obtain a group of linear equations to

1st Reading

Optimizing Data Intensive Flows for Networks on Chips

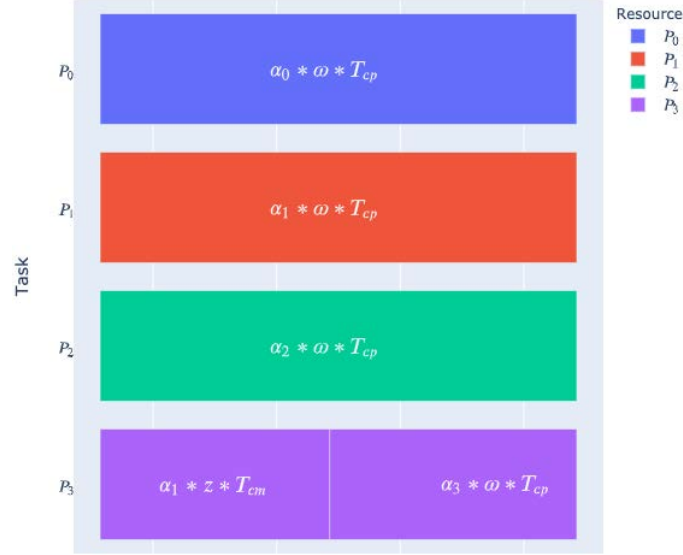


Fig. 3. The timing diagram for 2*2 mesh network with virtual cut-through and the root processor at P_0 .

- 1 find the fraction workload assigned to each processor α_i :

$$2 \quad \left\{ \begin{array}{l} \alpha_0 \omega T_{cp} = T_{f,m} \\ \alpha_1 \omega T_{cp} = T_{f,m} \\ \alpha_2 \omega T_{cp} = T_{f,m} \\ \alpha_1 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} \\ \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 = 1 \\ \sigma = \frac{z T_{cm}}{\omega T_{cp}} \\ 0 < \sigma < 1 \\ 0 < \alpha_0 \leq 1 \\ 0 \leq \alpha_1, \alpha_2, \alpha_3 < 1 \end{array} \right.$$

- 3 The group of equations are represented by the matrix form:

$$4 \quad \begin{bmatrix} 1 & 2 & 1 \\ 1 & -1 & 0 \\ 0 & \sigma & -1 & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad (1)$$

- 5 The matrix is represented as $A \times \alpha = b$. A is named as the **flow matrix**.
- 6 Because of the symmetry $\alpha_1 = \alpha_2$, the α_2 is not listed in the matrix equations. The
- 7 first row in flow matrix describes the number of cores on each hop with respect to
- 8 node 0.

- 9 • There is 1 core with 0 hop distance.

1st Reading

J. Zhang et al.

- 1 • There are 2 cores with 1 hop distance.
- 2 • There is 1 core with 2 hop distance.

3 Finally, the explicit solution is:

$$4 \quad \left\{ \begin{array}{l} \sigma = \frac{zT_{cm}}{\omega T_{cp}} \\ \alpha_0 = \frac{1}{4 - \sigma} \\ \alpha_1 = \frac{1}{4 - \sigma} \\ \alpha_3 = \frac{1 - \sigma}{4 - \sigma} \end{array} \right.$$

5 The performance result is illustrated:

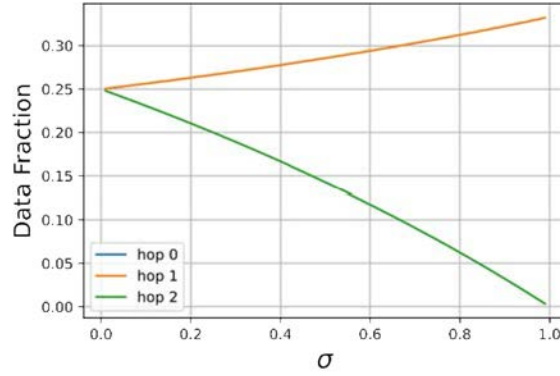


Fig. 4. 2*2 mesh network. $\alpha_0, \alpha_1, \alpha_2$ (upper curve), α_3 (lower curve) value curves.

6 In Fig. 4, the three processors P_0, P_1, P_2 have the same data fraction workload,
 7 so the curve of α_0, α_1 and α_2 coincide. The figure illustrates that as σ grows, the
 8 curve of α_3 drops. In other words, as the communication speed decreases, there is
 9 less data workload assigned to P_3 . Further, it means it will be economical to keep
 10 the load local on P_0, P_1, P_2 and don't distribute it, to other processors. Thus for
 11 slow communication $\alpha_0 = \alpha_1 = \alpha_2 = \frac{1}{3}$.

12 The equivalence inverse speed of a single processor is w_{eq} , that can replace the
 13 original network as

$$14 \quad \hat{T}_f = 1 * w_{eq} * T_{cp}$$

$$16 \quad w_{eq} = \alpha_0 * w$$

$$18 \quad Speedup = \frac{T_f}{\hat{T}_f} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = 4 - \sigma$$

For a fast communication ($\sigma \approx 0$), the speedup is 4 and the data fraction is about 0.25. The flow matrix concept is applicable to larger networks. The first row holds the number of nodes i hops from the source in the node $(0, i)$. The rest of the matrix has a practical structure [21]. Since $A \times \alpha = b$, one solves the linear flow matrix equation for the fraction of load assigned to each node processor α_i . See the next section and [20, 21] for more examples.

3. Multi-source Uniform Data Fraction

The single source assignment problem [20] has been studied. In this section, we focus on the general multi-source assignment problem [23, 19]. For each processor, we focus on the processors' geographical location P_i , and the data fraction α_i assigned. In addition, we assume that the data fractions are distributed uniformly. For example, in the case of a one unit workload and K data injection nodes, each data injection node is assigned $\frac{1}{K}$ of the workload. Regarding the data injection position relationship, we consider three different models :

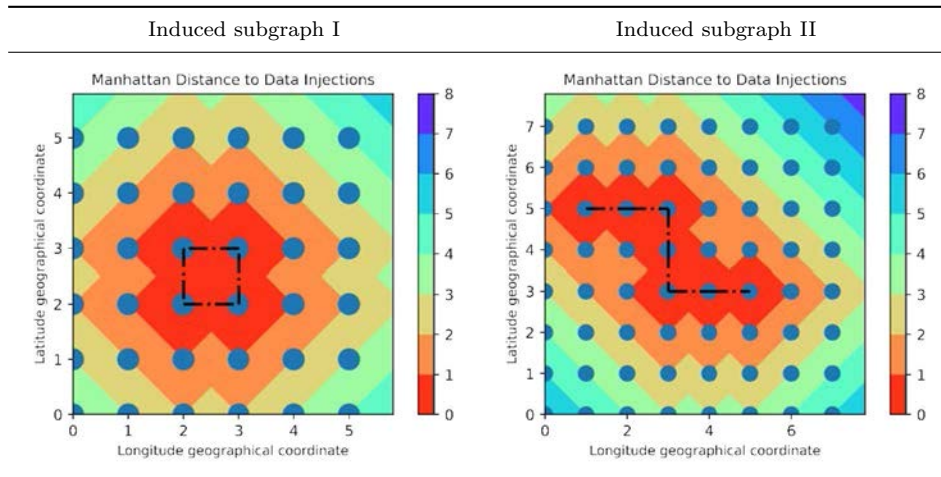
- (1) Data injection nodes consist of a connected induced subgraph G_L of G .
- (2) Data injection nodes don't connect with each other.
- (3) Some data injection nodes consist of some connected induced subgraphs and some are individual injection points.

3.1. Model I

In the scenario that the data injection positions consist of a connected induced subgraph of G , we use G_L to present it.

Table 1 illustrates two examples where the data injectors consist of connected induced subgraph of G . We propose a general algorithm framework to minimize the

Table 1. Data injectors consist of a connected induced subgraph of G .



1st Reading

J. Zhang et al.

1 makespan and present a quantitative model analysis utilizing the flow matrix [20].
 2 This algorithm is an extension of the processor equivalence's application in daisy
 3 chain [18], which is named as ***Equivalence Processor Scheduling Algorithm***
 4 (***EPSA***).

Algorithm 1 Equivalence Processor Scheduling Algorithm (EPSA)

Input: k data injection positions

Output: $m * n$ processor data fractions α_i

Collapse the data injection processors into one “big” equivalent processor [18].

Calculate $m * n$ processor's D_i .

Obtain the flow matrix A [20].

Calculate the determinant of flow matrix.

Calculate $m * n$ processors' data fraction α_i .

5 In term of the time complexity : according to the proof of [20] [21], we know the
 6 speedup is:

$$\hat{T}_f = 1 * w_{eq} * T_{cp}$$

$$w_{eq} = \alpha_0 * w$$

$$Speedup = \frac{T_f}{\hat{T}_f} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = |-\det A|$$

12 In addition, the time complexity of calculating the determinant is $O(r^3)$ with Gaus-
 13 sian elimination or LU decomposition. The r is the rank of matrix. The time com-
 14 plexity of calculating the flow matrix A is $O(m * n)$. And the total time complexity
 15 is $O(r^3 + m * n)$. Note that a critical step in the EPSA is to obtain the flow matrix
 16 given a graph. For example, connected subgraph I in Table 1 (1, 1)'s ***flow matrix***
 17 is:

$$\begin{bmatrix} 4 & 8 & 12 & 8 & 4 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & \sigma - 1 & 1 & 0 & 0 \\ 0 & \sigma - 1 & \sigma & 1 & 0 \\ 0 & \sigma - 1 & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (2)$$

19 The first row in flow matrix describes the number of cores on each D_i .

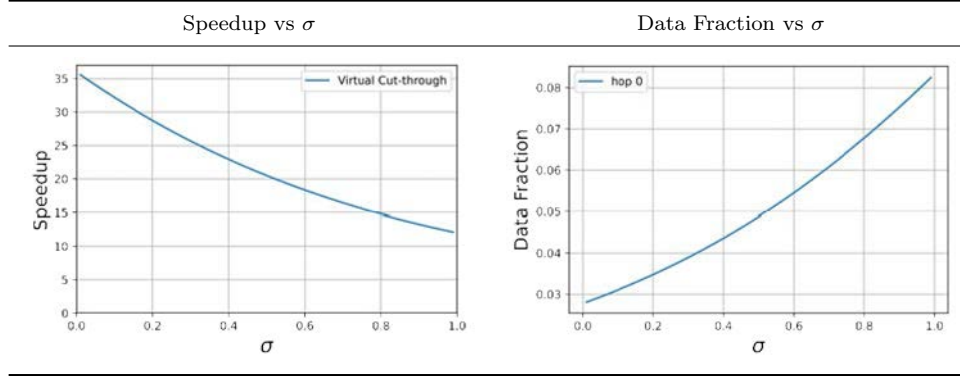
- 20 • There are 4 cores with 0 hop distance from the cluster.
- 21 • There are 8 cores with 1 hop distance from the cluster.
- 22 • There are 12 cores with 2 hop distance, and so on from the cluster.

23 Since each layer processor has the same data fraction, the size of the rows in the
 24 network yields the different data fractions.

1st Reading

Optimizing Data Intensive Flows for Networks on Chips

Table 2. Induced subgraph speedup vs σ and data fraction vs σ for Table 1 (1, 1).



We study the performance of the EPSA algorithm numerically. As shown in Table 2, the best performance occurs for value $\sigma \leq 0.05$ (σ is the ratio between the communication speed to the computation speed), for fast communication, which hits about 36 times speedup as all 36 processors in the network are engaged in processing networks. There are 12 processors at the initial stage, (four nodes are hop 0 and eight nodes are on hop 1). When $\sigma \approx 1$ (slow communication), in which, communication time equals computation time, the network achieves about 12 times speedup performance as only those 12 processors do processing. Note the $\sigma \approx 1$ scenario represents the slowest feasible communication. When the $\sigma > 1$, any distribution of the load yields worse performance than simply processing the load on injection nodes. In the Table 2, the α represents the task fraction processed by a 0 hop distance core. When the $\sigma \leq 0.05$, the fraction is about $\frac{1}{36}$ and when the $\sigma \approx 1$, the fraction is about $\frac{1}{12}$.

3.2. Model II

In the scenario that the data injection nodes don't form a whole connected subgraph of G , our objective is to propose a general heuristic algorithm framework to minimize the makespan. The seeds are induced subgraphs as well as "isolated" single injection nodes. For each Voronoi seed(s) there is a corresponding region consisting of all points closer to that seed(s) than any other. For the multi data injection nodes scenario, the intuitive algorithm is 'divide and conquer', which extends [23]'s graph partitioning algorithm to the Manhattan distance Voronoi diagram.

3.2.1. Manhattan distance Voronoi diagram algorithm

Similar to the EPSA algorithm, the objective here is to find proper data fractions which minimize the makespan. This problem can be formulated into a linear programming problem whose definition is given in the following.

J. Zhang et al.

1 Linear Program

2

$$3 \quad \min \quad \hat{T}_f \quad (3)$$

$$4 \quad \text{s.t.} \quad \sum_{j=0}^{k-1} \sum_{i=0}^{S_j-1} \hat{\alpha}_{i,j} = 1 \quad (4)$$

$$5 \quad 1 \geq \hat{\alpha}_{i,j} \geq 0, \quad (5)$$

$$6 \quad T_{f,i} = T_{f,0} \quad (6)$$

$$7 \quad \sum_{n=1}^{n=j-1} \hat{\alpha}_{i,n} z T_{cm} + \hat{\alpha}_{i,j-1} w T_{cp} = \hat{T}_{f,k}, \quad 1 \leq j \leq L \quad (7)$$

- 8 • Equation (3) is the objective function, minimizing the makespan. \hat{T}_f is the
- 9 finish time of the whole processor network. Here \hat{T}_f is equal to $w_{eq} T_{cp}$.
- 10 • Equation (4) means that all total fraction of all processors is unit 1. Here $\hat{\alpha}_{j,i}$
- 11 is the load fraction assigned to i th processor of j th cell.
- 12 • Equation (5) means that in each cell, each processor's data fraction is nonneg-
- 13 ative.
- 14 • Equation (6) means that in each cell, each processor has the same finish time.
- 15 • Equation (7) means that in each cell, the j th layer's finish time consists of
- 16 previous $(j - 1)$ layers' transmitting time and j th computation time.

17 We further propose an algorithm, named ***Manhattan Distance Voronoi Diagram Algorithm*** to solve this problem:

Algorithm 2 Manhattan Distance Voronoi Diagram Algorithm (MDVDA)

Input: k data injection positions

Output: $m * n$ processor data fractions α_j .

Collapse the connected data injectors into some “big” equivalent processors and the number of clusters is k .

Calculate k Voronoi cells with Manhattan distance. The k injectors are the Voronoi “seeds”.

Calculate the flow matrix A_i .

Calculate the determinant of flow matrix A_i .

Calculate $m * n$ processors' data fraction α_j .

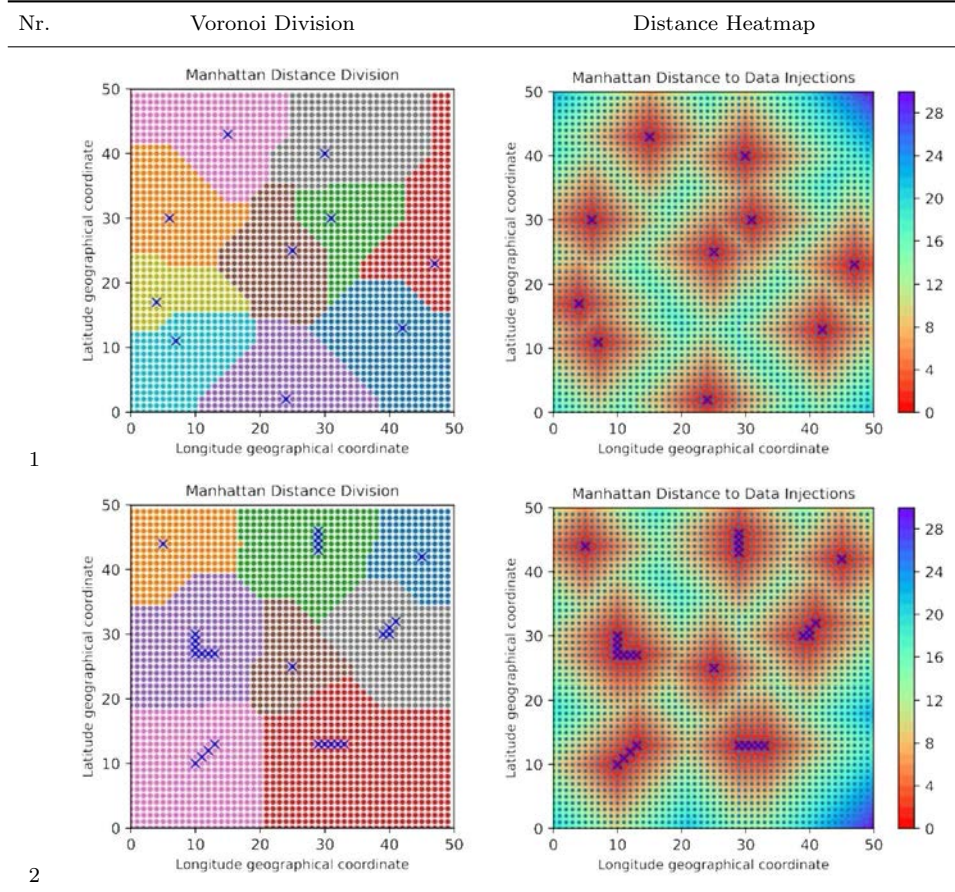
18

19 The time complexity is the same as with EPSA. Manhattan distance Voronoi
 20 diagram division Table 3 shows 10 and 8 Voronoi cells division scenario. The first
 21 row shows the 10 individual data injectors mesh division result and the distance
 22 heat-map which shows the core processor to its nearest data injector's Manhat-
 tan distance. In the second row, we consider another more general scenario. The

1st Reading

Optimizing Data Intensive Flows for Networks on Chips

Table 3. Model 2: Voronoi division and distance heatmap.



- 1 data injectors are not only individual cores but also induced sub-graphs injectors.
- 2 Fig. (2,2) shows the Manhattan distance to its nearest injectors. Each division will
- 3 handle the corresponding task from the data injector.

3.3. Model III

- 5 According to a general situation, if there are some nodes consisting of introduced
- 6 sub-graphs and other nodes are individual seeds, our objective to minimize the
- 7 makespan and save processors. After investigation, we find that the makespan de-
- 8 pends on the bottleneck cell makespan. In other words, if other cells contain more
- 9 processors than the bottleneck cell, it does not help to minimize the makespan.
- 10 We propose an algorithm named Reduced Manhattan Distance Voronoi Diagram
- 11 Algorithm (RMDVDA) to tackle this situation.

J. Zhang et al.

Algorithm 3 Reduced Manhattan Distance Voronoi Diagram Algorithm (RMD-VDA)

Input: \hat{k} data injection positions

Output: $m * n$ processor data fractions α_i .

Collapse the connected data injection processors into some “big” equivalent processors and the number of cluster is k .

Calculate k constrained Manhattan distance Voronoi cells [24].

Calculate the constrained Voronoi cells’ radius R_i .

Calculate the constrained Voronoi cells’ flow matrixes A_i .

Calculate the constrained Voronoi cells’ speedup Sp_i .

Set the $depth_{min} = \min(Sp_i)$ ’s radius.

Calculate k reduced constrained Voronoi cells by setting the $depth_i = depth_{min}$ in each Voronoi cell.

Calculate Voronoi cell’s flow matrix \hat{A}_i .

Calculate the determinant of flow matrix \hat{A}_i .

Calculate $m * n$ processors’ data fraction α_i .

1 In terms of the time complexity, it’s the same with EPSA.

- 2 • Voronoi speedup curves in Table 4(3,1) shows that for $\sigma < 0.1$, the ratio
- 3 $\max(\frac{Sp_{max}}{Sp_{min}}) = \frac{293}{133} \approx 2.2$.
- 4 • Reduced Voronoi speedup curves in Table 4(4,1) show that for $\sigma < 0.1$, the
- 5 ratio is $\max(\frac{Sp_{max}}{Sp_{min}}) = \frac{153}{133} \approx 1.13$.

6 To explain, the whole network makespan depends on the max makespan of each

7 core division. So if the ratio of $\max(\frac{Sp_{max}}{Sp_{min}})$ is more balanced, in other words, the

8 curves are closer to each other, it is better, for example, Table 4 (3,1), (4,1) and

9 (3,2), (4,2). The divisions in Table 4(1,1) and Table 3(1,1) display that 10 cells’

10 equivalence computation is more balanced than the initial setting, and the whole

11 cluster finishes processing load within the same time but with only $1462/2500 \approx$

12 58.5% of the processors.

13 Table 5 shows simulation experiments. A boxplot is a standardized way of dis-

14 playing the dataset based on a five-number summary: the minimum, the maximum,

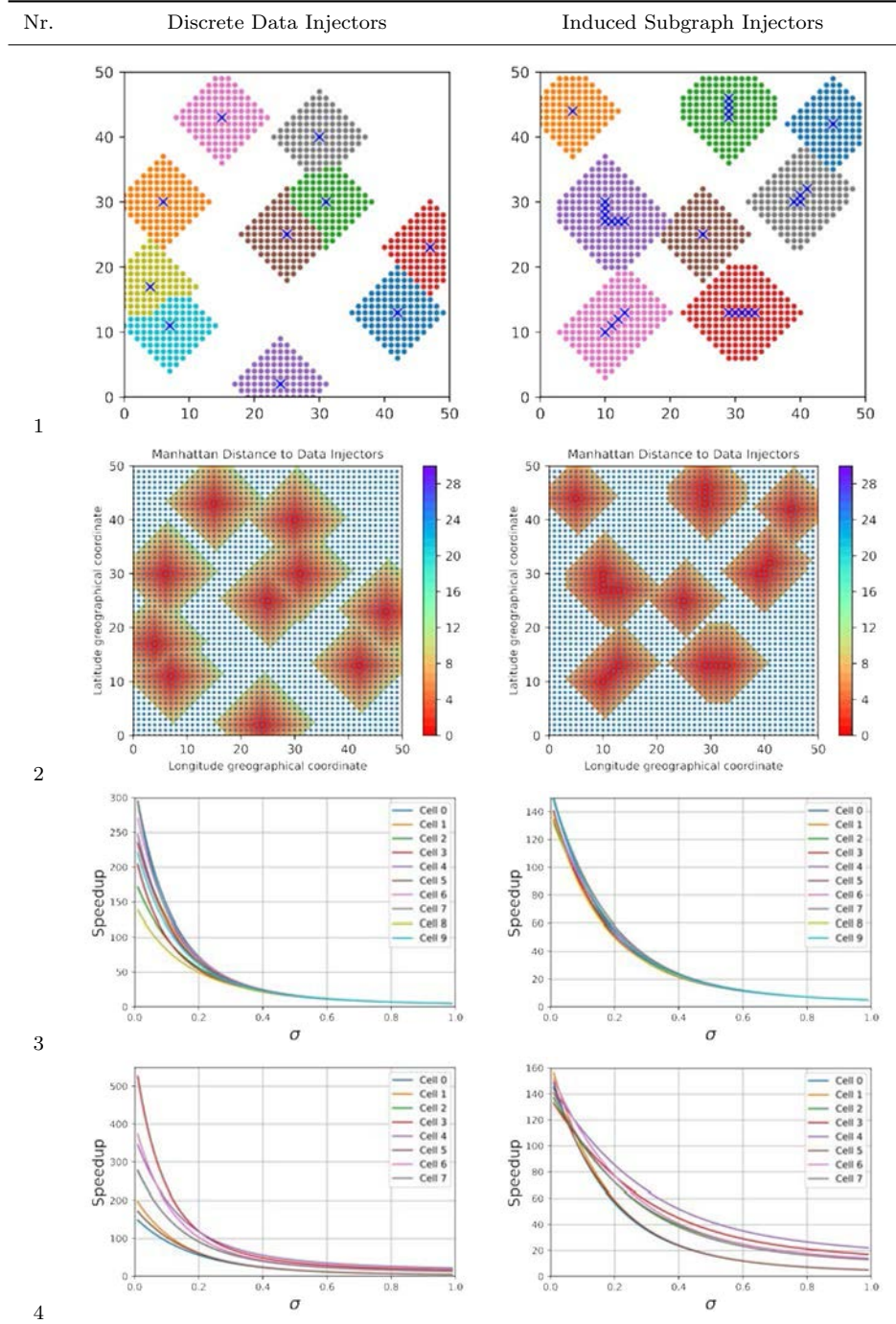
15 the sample median, and the first and third quartile.

- 16 • Table 5 (1,1) in Table 5 shows the initial Voronoi division. The regular mesh
- 17 grid edge core number is 50 and there are 10 random data injectors. The work-
- 18 load fraction is uniform, so it is $\frac{1}{10}$.
- 19 • Table 5 (1,2) describes after 2000 rounds of simulation in each σ , as the number
- 20 of data injectors grows from 10 to 20,
- 21 – In $\sigma = 0.1$ scenario, our algorithm is robust and it saves about 40% cores
- 22 in experiments.

1st Reading

Optimizing Data Intensive Flows for Networks on Chips

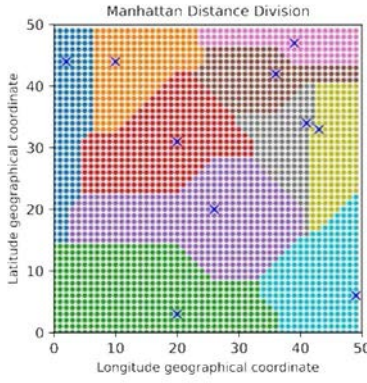
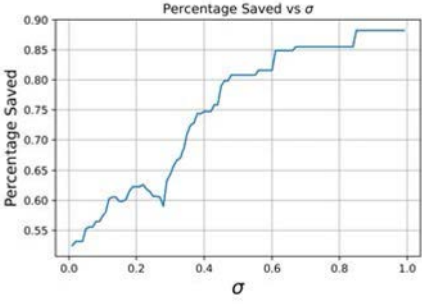
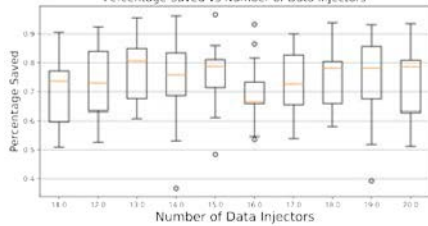
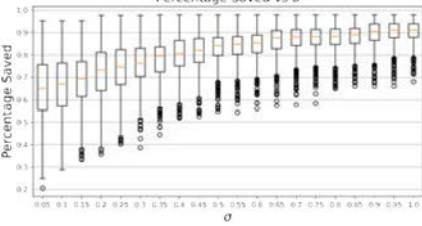
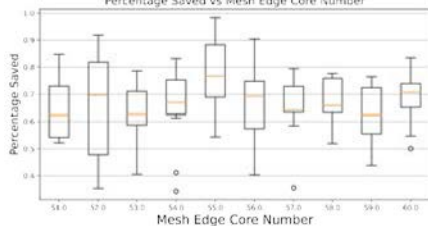
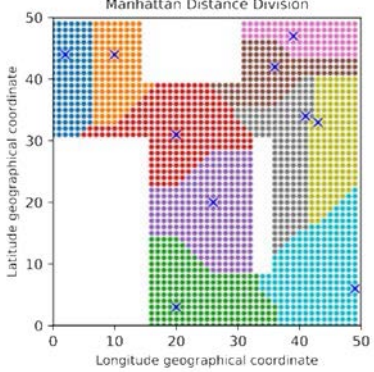
Table 4. Model III Voronoi division and reduced Manhattan distance heatmap.



1st Reading

J. Zhang et al.

Table 5. Simulation statistics result: Voronoi division, percentage saved vs σ , percentage saved vs number of data injectors, percentage saved vs σ .

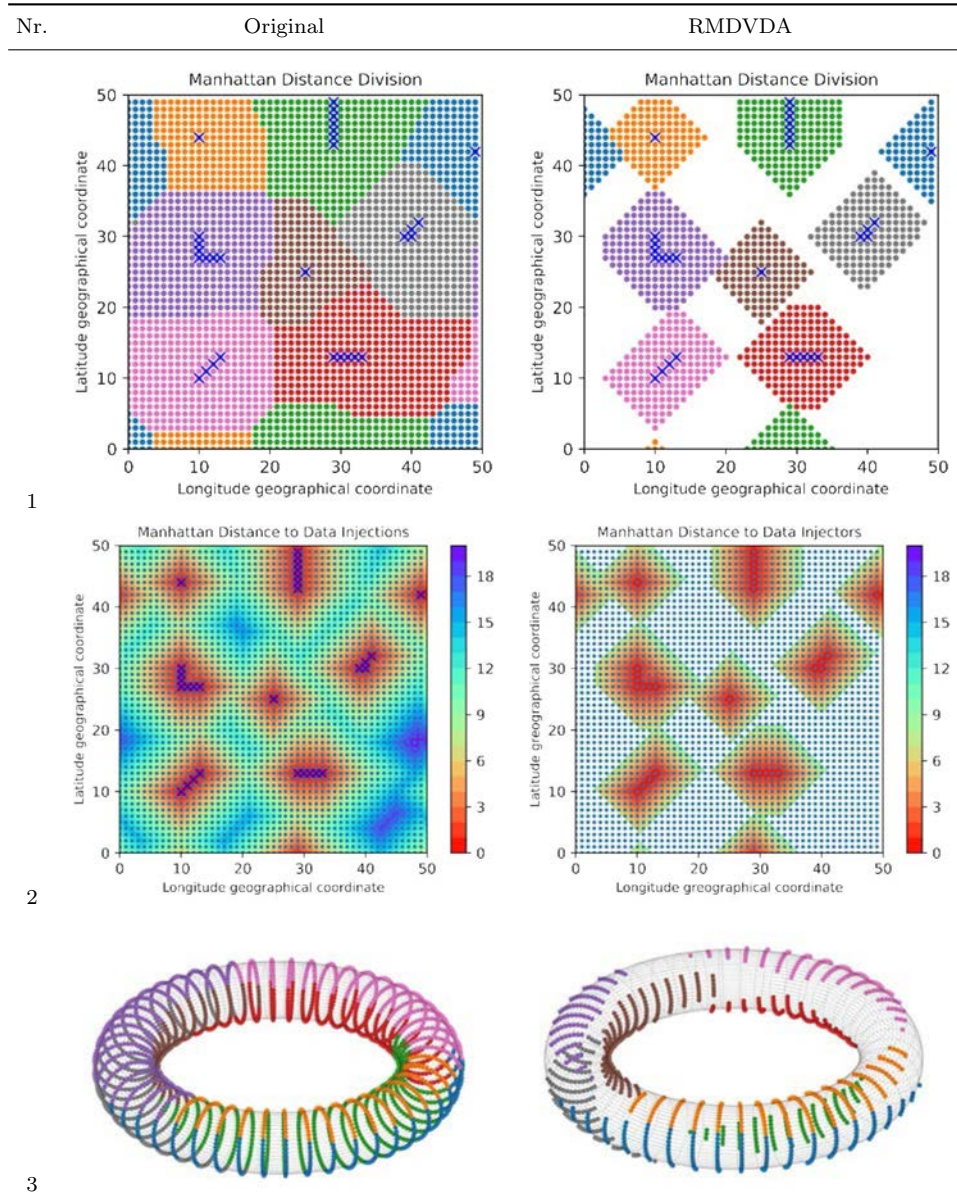
Nr.	Division	Simulation Result
1		
2		
3		

- 1 – In the $\sigma > 0.5$, it saves over 80% of the cores. It means the algorithm
- 2 keeps more data to process in local area instead of broadcasting it to
- 3 further cores.
- 4 • Considering the edge core number is 50 and $\sigma = 0.1$ scenario, we adjust the data
- 5 injector number from 11 to 20. Table 5 (2, 1) boxplot describes the simulation
- 6 result. For example, the data injector number is 11 and we randomly place the
- 7 11 data injectors to the mesh 1000 times and summarize the simulation result

1st Reading

Optimizing Data Intensive Flows for Networks on Chips

Table 6. Model 3: Torus Voronoi division and distance heatmap.



in the boxplot. The minimum is about 51%, the first quartile is over 60%, the median is 75%, the third quartile is 77% and the maximum is 90%. The first quartile is over 50%, which means in 1000 times simulation experiments, the 75% tests achieve 50% percentage saved performance. In sum, the second quartile of different injectors scenario is over 50%, so the RMDVDA is robust.

J. Zhang et al.

- 1 • Considering the edge core number is 50 and the data injector number is 10.
 2 Table 5 (2,2) shows the simulation result as the $0 < \sigma < 1$. For example, in
 3 $\sigma = 0.1$, after 1000 experiments, the boxplot shows the minimum percentage
 4 saved is 29%, the first quartile is 57%, the median is 66%, the third quartile is
 5 76% and the maximum is 95%. The circles are the outliers. In sum, the median
 6 of each boxplot increases as the σ grows and the value is over 60%, so the
 7 RMDVDA is robust.
- 8 • Considering the data injectors number is 10 and the $\sigma = 0.1$, Table 5 (3,1)
 9 shows the simulation summary as the mesh grid edge from 51 to 60. For ex-
 10 ample, the edge core number is 53, we randomly place 10 data injectors in
 11 the mesh 1000 times and the boxplot shows the minimum percentage saved is
 12 40%, the first quartile is 59%, the median is 62%, the third quartile is 70% and
 13 the maximum is 79%. In sum, the median of each setting is over 50%, so the
 14 RMDVDA is robust.
- 15 • Table 5 (3,2) shows that RMDVDA extend to irregular graph successfully,
 16 which handles the graph with holes.

17 We also can extend our method to torus mesh networking (Table 6).

18 4. Thermal Management of NoC

19 The thermal management of NoC is a complex problem. Recall power is energy per
 20 time. If these algorithms use fewer cores, the active cores' overall power usage may
 21 not change due to increased computational load. Moreover, because the flow matrix
 22 algorithm presented takes communication and computation into account, the whole
 23 divisible load's total execution time may have less than a proportional increase in
 24 using fewer cores (energy is power times execution time for a load). This depends
 25 on the value of σ . Using fewer cores will result in less fixed energy/power costs.
 26 Finally, using fewer cores allows more flexibility in hot spot management. Using
 27 fewer cores may also be an advantage in freeing up cores for other jobs, some of
 28 which may require a free core(s).

29 5. Discussion on Load Sharing

30 5.1. Question

31 Assume in a homogeneous network, an embedded tree is distributing load in a
 32 network. For a binary tree node, one may have a link with the data rate of $1/z$, and
 33 the outgoing total data rate is $2/z$. It, on the surface, violates the conservation of
 34 data entering and leaving a node.

35 5.2. Explanation

36 The issue is complex as for some distribution policies,

- 1 • Firstly, not all outgoing links of a node may be active at the same time (say
2 under sequential load distribution).
- 3 • Secondly, suppose that a store and forward policy is used in a load distribution
4 policy. In that case, all data is present at the (possibly intermediate) root
5 node before forwarding to output links occurs, so that there is no mismatch of
6 data transfer. What is present in the simulations of this paper is a mixture of
7 virtual cut through (at least some of the children nodes of a parent node start
8 computing as they receive load at time zero) and store and forward (as described
9 in this bullet). Because of the store and forward aspect, the simulations in this
10 paper do not have the issue of a mismatch of data rate.
- 11 • Finally, usually less data may leave a node than enters it (the node keeps, i.e.,
12 some for processing) so that the outgoing links from a node are not as active
13 as the incoming link to a node.

14 Still, if one chooses an overall virtual load distribution tree technique what can
15 we do to mitigate this problem? To mitigate the problem, in the virtual distribution
16 links, one needs two constraints:

- 17 • The data rate of each load distribution link is less than the physical capacity
18 of each link.
- 19 • A node's effective virtual input data rate matches the node's effective total
20 virtual output data rate.

21 This issue was addressed in the earlier divisible load theory literature (often using
22 heuristic methods). See [8, 25–27].

23 6. Conclusion

24 This work is significant in proposing the use of established Voronoi diagram tech-
25 niques for finding efficient NoC flow distributions. It is novel in the use of optimal
26 linear solution technique based on the flow matrix for finding flow distributions
27 within individual clusters. Savings in the number of processors used are impressive
28 and bode well for minimizing chip power consumption.

29 Acknowledgement

30 The authors wish to thank Prof. Emre Salman and Prof. Maciej Drozdowski for
31 useful discussions. We also thank the anonymous reviewers' very useful comments.

32 References

- 33 [1] T. G. Robertazzi, *Introduction to Computer Networking* (Springer Science, 2017).
- 34 [2] L. Benini and G. De Micheli, Networks on chips: A new SoC paradigm, *Computer*
35 **35**(1) (2002) 70–78.
- 36 [3] A. Ganguly, K. Chang, S. Deb, P. P. Pande, B. Belzer and C. Teuscher, Scalable hy-
37 brid wireless network-on-chip architectures for multicore systems, *IEEE Transactions*
38 *on Computers* **60**(10) (2011) 1485–1502.

J. Zhang et al.

- 1 [4] K. Tatas, K. Siozios, D. Soudris and A. Jantsch, *Designing 2D and 3D Network-on-Chip Architectures* (Springer, 2016).
- 2 [5] E. Krevat, J. G. Castaños and J. E. Moreira, Job scheduling for the BlueGene/L system, in *Workshop on Job Scheduling Strategies for Parallel Processing* (Springer, 2002), pp. 38–54.
- 3 [6] V. Bharadwaj, D. Ghose and T. G. Robertazzi, Divisible load theory: A new paradigm for load scheduling in distributed systems, *Cluster Computing* **6**(1) (2003) 7–17.
- 4 [7] V. Bharadwaj, D. Ghose, V. Mani and T. G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems* (John Wiley & Sons, 1996).
- 5 [8] M. Drozdowski, *Scheduling for Parallel Processing* (Springer, 2009).
- 6 [9] H. Casanova, A. Legrand and Y. Robert, *Parallel Algorithms* (Chapman and Hall/CRC, 2008).
- 7 [10] J. Sohn and T. G. Robertazzi, Optimal divisible job load sharing for bus networks, *IEEE Transactions on Aerospace and Electronic Systems* **32**(1) (1996) 34–40.
- 8 [11] H. M. Wong, D. Yu, B. Veeravalli and T. G. Robertazzi, Data intensive grid scheduling: Multiple sources with capacity constraints, in *Fifteenth IASTED International Conference on Parallel and Distributed Computing and Systems*, **1** (2003), pp. 7–11.
- 9 [12] L. Marchal, Y. Yang, H. Casanova and Y. Robert, A realistic network/application model for scheduling divisible loads on large-scale platforms, in *19th IEEE International Parallel and Distributed Processing Symposium* (IEEE, 2005), 10 pp.
- 10 [13] T. Lammie and G. Thomas, A linear daisy chain with two divisible load sources, *Conference Information Science and Systems* (2005).
- 11 [14] T. G. Robertazzi and D. Yu, Multi-source grid scheduling for divisible loads, in *Proceedings of the 40th Annual Conference on Information Sciences and Systems* (2006), pp. 188–191.
- 12 [15] A. Okabe, B. Boots, K. Sugihara and S. N. Chiu, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams* (John Wiley & Sons, 2009).
- 13 [16] I. Stojmenovic, A. P. Ruhil and D. Lobiyal, Voronoi diagram and convex hull based geocasting and routing in wireless networks, *Wireless Communications and Mobile Computing* **6**(2) (2006) 247–258.
- 14 [17] S. Meguerdichian, F. Koushanfar, G. Qu and M. Potkonjak, Exposure in wireless ad-hoc sensor networks, in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking* (ACM, 2001), pp. 139–150.
- 15 [18] T. G. Robertazzi, Processor equivalence for daisy chain load sharing processors, *IEEE Transactions on Aerospace and Electronic Systems* **29**(4) (1993) 1216–1221.
- 16 [19] X. Liu, H. Zhao and X. Li, Scheduling divisible workloads from multiple sources in linear daisy chain networks, in *PDPTA* (2007), pp. 528–534.
- 17 [20] J. Zhang, Y. Liu, S. Li and T. G. Robertazzi, Optimizing data intensive flows for networks on chips, arXiv preprint arXiv:1812.07183 (2018).
- 18 [21] J. Zhang, Data distribution equivalence for data intensive interconnection networks, PhD dissertation, Dept. of Applied Mathematics and Statistics, Stony Brook University (2018).
- 19 [22] J. T. Hung and T. G. Robertazzi, Switching in sequential tree networks, *IEEE Transactions on Aerospace and Electronic Systems* **40**(3) (2004) 968–982.
- 20 [23] J. Jia, B. Veeravalli and J. Weissman, Scheduling multisource divisible loads on arbitrary networks, *IEEE Transactions on Parallel and Distributed Systems* **21**(4) (2010) 520–531.
- 21 [24] F. Chin and C. A. Wang, Finding the constrained delaunay triangulation and constrained Voronoi diagram of a simple polygon in linear time, *SIAM Journal on Computing* **28**(2) (1998) 471–486.

1st Reading

Optimizing Data Intensive Flows for Networks on Chips

- 1 [25] J. Błażewicz and M. Drozdowski, The performance limits of a two dimensional
2 network of load-sharing processors, *Foundations of Computing and Decision Sciences*
3 **21**(1) (1996) 3–15.
- 4 [26] M. Drozdowski and P. Wolniewicz, Performance limits of divisible load processing
5 in systems with limited communication buffers, *Journal of Parallel and Distributed*
6 *Computing* **64**(8) (2004) 960–973.
- 7 [27] W. Głażek, A multistage load distribution strategy for three-dimensional meshes,
8 *Cluster Computing* **6**(1) (2003) 31–39.